BROWN UNIVERSITY
Department of Computer Science
Master's Project
CS-96-M6

"Learning Dynamical Systems Using Hidden Markov Models"

by

Sonia Leach

Learning Dynamical Systems Using Hidden Markov Models

Sonia Leach

Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University

May 1996

Leslie Pack Kaelbling

Professor Leslie Pack Kaelbling
Advisor

Learning Dynamical Systems Using Hidden Markov Models [1]

Sonia Leach
Department of Computer Science
Brown University
115 Waterman Street
Providence, RI 02912, USA
Phone: (401) 863-7687
Fax: (401) 863-7657
Email: sml@cs.brown.edu

In this paper, we address the problem of learning models for complex processes under the assumption that the processes can be represented within the hidden Markov model (HMM) framework. Toward that aim, we investigate the strengths and weaknesses of two competing algorithms for learning HMMs: Baum-Welch and Bayesian Model Merging. We offer insight into the reasons for the success or failure of each algorithm, especially through empirical trials, in several domains. Our experiments support the conclusion that Bayesian Model Merging suffers a number of disadvantages which suggest Baum-Welch be preferred for learning a particular class of HMMs.

Keywords: dynamical systems, learning, hidden Markov models, Baum-Welch, Bayesian Model Merging

[1]Preliminary versions appear in joint work with Thomas Dean, Leslie Pack Kaelbling, Kee-Eung Kim, and Hagit Shatkay cited in the references [Dean *et al.*, 1996a, Dean *et al.*, 1996b]

1 Introduction

Real-world processes typically generate data in the form of a sequence of observable outputs of the system. Often the data suggests that the system exhibits sequentially changing behavior, where a period of one distinct behavior is followed by a period of another distinct behavior. These periods are commonly referred to as states and we are interested in how the system evolves from one state to another. Specifically, we are interested in characterizing the behavior of the system in the form of a mathematical model, which we can then use later for prediction or explanation. Given a good enough model, we could also simulate the process and learn from the simulation, without the expense of gathering data from the real source.

As an example, a physician might use a model of the blood sugar level in a diabetic patient to investigate how the patient would respond to varying levels of insulin injection. Or a computer systems administrator might use a model of network use to determine how an investment in new technology would affect network performance. Alternatively, speech recognition systems might use models to distinguish between two word pronunciations, based on the behavior of the signal. Similar examples can be found in industrial process control, transportation planning, financial forecasting, protein classification and alignment, and a host of other applications.

An important aspect of modelling complex systems is to discover structure in the underlying process. By this we mean, discovering the states and the dynamics governing the transitions between those periods. We assume that we can observe the system at discrete time periods, and that our observations are also discrete. For example, suppose we are trying to model the behavior of a robot which can be in one of three rooms. The robot travels from room 1 to room 2, then to room 3, and returns to room 1. However, its path is sometimes blocked by a door separating room 1 and room 2. We might describe the states of the system at any point in time in terms of which room the robot is in and whether or not the door is closed, for a total of 6 states. However, if we note that the robot moving from room 2 to room 3 is completely independent of whether the door is open or closed, then we can group states based on this information and use only 4 distinct states. Knowing how the door affects the location of the robot enables a more efficient and compact representation of the process. Suppose further that we cannot observe the status of the door, so we do not know which state the robot is in at any point in time. We might still be able to infer the connection between the robot's location and the status of the door from the observation sequence over time. We might also be able to infer the door's status if we observe whether the robot remains in room 1 for any period of time. In general, our hope is that such structure is discernible from the outputs of a process, even in cases where the dynamics of the underlying process are unknown.

The theory of hidden Markov models (HMMs) presents a nice framework for modelling partially observable processes. Here the system dynamics are described in terms of a set of states, a set of observations, and probability distributions governing the initial states, state transitions, and observations at the states. The distribution over observations at each state takes into account the fact that some aspects of the state are hidden, so the same observation may be made at different states. Learning the distributions, or parameters, for a particular model of a process requires searching in the space of all parameter settings, guided by some performance criterion. Typically, the performance criterion is measured in terms of the difference between the learned model and the target model (either the actual model or proxy in the form of the available data).

The learning algorithms we consider take as input sequences of observations generated by the process and return a model in the form of the aforementioned distributions. In particular, we examine two prominent algorithms for learning HMMs: the Baum-Welch algorithm [Baum *et al.*, 1970] and the Bayesian Model Merging algorithm [Stolcke and Omohundro, 1993]. The Baum Welch (BW) procedure assumes the number of states in the model is fixed, and proceeds to adjust the model parameters, whereas the Bayesian Model Merging (MM) approach tries to induce both the model topology and parameters. BW has proven successful in many applications and its strengths and caveats are well known. MM is a more recent algorithm so little is known about its applicability to a wider range of problems. The scarcity of literature on MM argues for further evaluation of its performance. Hence, our discussion of the two algorithms will be biased toward MM.

In this paper, we address the problem of learning models for complex processes under the assumption that the processes can be represented within the hidden Markov model framework. Toward that aim, we investigate the strengths and weaknesses of two competing algorithms for learning HMMs, especially through empirical trials on several domains. Such a study will not only help us to understand how these specific algorithms work and to compare their relative performance on a variety of problems, it might also give insight into the properties of a domain that make learning the structure harder or easier in general.

The remainder of this paper is organized as follows. Section 2 provides the necessary definition of a hidden Markov model and details about each of the learning algorithms. Section 3 contains empirical results of applying the algorithms to examples of regular languages and the robot domain mentioned above. Our experiments provide insight into when and why these algorithms succeed or fail. Our results show that Bayesian Model Merging is not such a desirable alternative to Baum-Welch as earlier reports would suggest. Lastly, Section 4 offers further discussion and conclusions.

2 Hidden Markov Models

Hidden Markov models (HMMs) present a general statistical framework for representing stochastic processes where the state of the system is unknown or hidden from observation. The general problem of learning HMMs is not known to be in the class of problems with polynomial time solutions [Abe and Warmuth, 1992]. However, learning algorithms exist which provide approximate solutions and we experiment with two in particular. The remainder of this section is dedicated to a brief description of each algorithm. We first give the formal definition of an HMM which we adopt throughout the paper.

2.1 Definition of an HMM

We assume a finite number of states and each state emits an output symbol from a finite alphabet (this is equivalent to saying an observation is made at each state). The output symbols provide indirect information about the underlying hidden state. The process governing state transitions is assumed to be Markovian, *i.e.,* the current state depends only on the immediate predecessor. Formally, we define an HMM as the following:

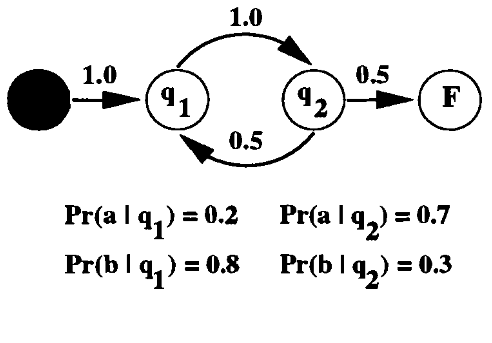
- A finite set of states $q_1, \dots, q_n \in \mathcal{Q}$.
- A finite set of observations $v_1, \dots, v_m \in \mathcal{V}$.
- A set of transition probabilities $A = \{a_{i,j}\}$, where $a_{i,j} = \Pr(S_{t+1} = q_j | S_t = q_i)$ is the probability of transitioning from state q_i to state q_j and S_t denotes the state at time t. If S_t does not refer to any specific q_i, we sometimes write $a_{S_t,S_{t+1}}$.
- A set of observation probabilities $B = \{b_{i,k}\}$, where $b_{i,k} = \Pr(O_t = v_k | S_t = q_i)$ is the probability of emitting observation v_k from state q_i and O_t denotes the observation at time t. We often refer to $b_{i,j}$ as *output* or *emission* probabilities. If S_t or O_t do not refer to any specific q_i or v_k, we sometimes write b_{S_t,O_t}.
- An initial state distribution $\pi = \{\pi_i\}$, where $\pi_i = \Pr(S_1 = q_i)$ is the probability of starting in state q_i.

An HMM is said to generate an observation sequence $O = O_1 O_2 \dots O_l$ of length l if and only if there is a state sequence, or path, $S_1 S_2 \dots S_l$ with non-zero probability, such that S_t outputs O_t and transitions to S_{t+1} with non-zero probability, for all $t = 1, \dots, l$ (no transition is made from the state S_l). The probability of a path $S_1 S_2 \dots S_l$ (relative to O) is the product of all transition and output probabilities along it, weighted by the probability of starting in state S_1. The conditional probability $\Pr(O|M)$ of an observation sequence O given an HMM M is computed as the sum of the probabilities of all paths that generate O:

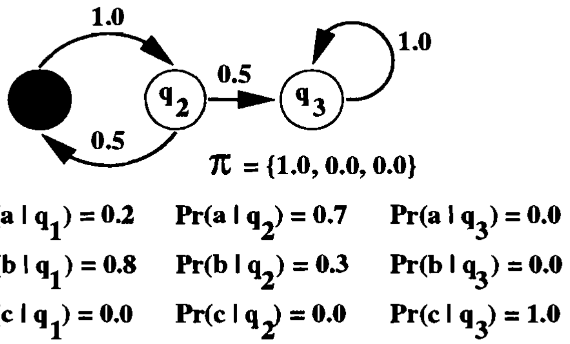
$$\Pr(O|M) = \sum_{S_1 \dots S_l} \pi_{S_1} b_{S_1,O_1} a_{S_1,S_2} \dots b_{S_{l-1},O_{l-1}} a_{S_{l-1},S_l} b_{S_l,O_l}$$

The definition used by [Stolcke and Omohundro, 1993, Stolcke and Omohundro, 1994] for the Bayesian Model Merging algorithm includes two special states: an initial state I which occurs at the beginning of any state sequence (hence π is unnecessary), and a final state F at the end of a state sequence. Neither of these states can occur anywhere else in the state sequence, nor do they emit output symbols. Generally, we assume $I, F \notin \mathcal{Q}$. Note that such a model can be captured by the above definition with appropriate modifications. This equivalence is important when we are discussing the learning algorithms, since the two approaches assume different, yet equivalent, definitions.

For example, Figure 1 illustrates the translation of the HMM, with a single start state I and single final state F, to an equivalent model according to the definition above. We adopt the convention that numbers above the arcs between states are the transition probabilities, and the initial state is highlighted in blue. From Figure 1a, the transition probability distribution of state I becomes the initial state distribution π and the final state F becomes state q_3 in Figure 1b. Also, the symbol c is added to the observation alphabet as a special symbol to indicate the end of an acceptable observation sequence, and the observation probabilities for each state are modified accordingly. The HMMs are equivalent since any observation sequence O generated by the HMM of Figure 1a has a probability p if and only if the observation sequence O', formed by concatenating O with any number of the symbol c, is generated by the HMM of Figure 1b with probability p.



(a)



(b)

Figure 1: Equivalence of Definitions

2.2 Learning Algorithms

In this paper, we examine two learning algorithms: Baum-Welch and Bayesian Model Merging. We assume the learner is given a sequence of observations and infers a model of the underlying dynamics governing the process which generated the data. In both cases, we adopt a Bayesian approach to learning, whereby we are interested in finding the model M that is most likely given the data O, *i.e.,* finding M maximizing $\Pr(M|O)$. By Bayes' rule, we have $\Pr(M|O) \propto \Pr(M)\Pr(O|M)$, where the term $\Pr(M)$ allows us to bias the search over models, and the term $\Pr(O|M)$, referred to as the sample likelihood, expresses how closely the model fits the data.

Below, we give a brief overview of each algorithm, presenting only the relevant ideas, and refer the reader to the appropriate references for more detailed descriptions. The key differences in the algorithms are in what features of the model the algorithm learns, how data from the target process is used in the learning, and how the Bayesian perspective plays a role. The Baum-Welch algorithm assumes a uniform prior $\Pr(M)$ over models of a fixed size, and begins learning from a *random* initial model with a *fixed* number of states. The algorithm then uses the data to learn that model's *parameters.* In contrast, the Bayesian Model Merging algorithm begins learning from a model *specific to the data*, and adjusts *both* the parameters of the model and its topology, using $\Pr(M)$ in its search to introduce a bias toward smaller models.

2.2.1 Baum-Welch

Baum-Welch [Baum *et al.*, 1970] (BW) is a special case of the expectation maximization (EM) algorithm of Dempster *et al.* [1977]. We use a variant of the algorithm described in Rabiner [Rabiner, 1989].

The goal of the Baum-Welch algorithm is to learn the probability distributions A, B and π for an unknown process, under the assumption that the process can be modeled as an HMM. Note that the distributions A and B can be thought of as matrices whose rows are the distributions for each state. The algorithm takes as its input a set of data sequences, O, generated by the unknown process, and the number of states and observations in the expected HMM. It starts by assigning some initial probability distribution (usually random) to each row of the matrices A and B as well as to π.

It applies a version of the EM algorithm to adjust these initial distributions by collecting expected sufficient statistics from the data in O for each of the model parameters. The algorithm iteratively adjusts all the probabilities, until a fixed point is reached. At this point, updating the matrices does not significantly change them. The resulting matrices form the model M' which induces a probability distribution over data sequences. The model M' is guaranteed to *locally* maximize the probability $\Pr(O|M) = \prod_{o \in O} \Pr(o|M)$ over all HMMs M with the same number of states. Note that this is equivalent to maximizing $\Pr(M|O)$ since BW assumes a uniform prior over models of a fixed size. However, the EM algorithm *does not* guarantee to find the model that globally maximizes $\Pr(O|M)$. In fact, we can only be certain that the learned model M' is no worse than the arbitrary model we started with. The choice of the initial model determines how close to the global maximum the learned model will be.

2.2.2 Bayesian Model Merging

Bayesian Model Merging (MM) is a technique developed by Stolcke and Omohundro [1993]. Unlike the Baum-Welch algorithm, which estimates the parameters of the model assuming a fixed number of states, MM induces both the model parameters and the model topology. The algorithm begins with the most specific model consistent with the training data and generalizes by successively merging states. The criterion for selecting which states to merge and when to halt the merging process is governed by the Bayesian posterior probability of the model.

The MM algorithm begins by constructing a model that simply replicates the data. There is a start state with as many outgoing transitions as there are sequences, and each sequence is represented by a unique path with one state per output symbol. The transitions from the start state are uniformly distributed, and a single transition from each of the interior states has probability one. The algorithm then attempts to learn the model by generalizing from the data. This is accomplished by merging pairs of states, effectively collapsing portions of the model with similar substructure. Intuitively, the initial model can be viewed as the result of "unrolling" the state sequences that were taken when generating the samples from the real model. The merging process simply attempts to undo the process and return the consolidated model.

The MM approach is best illustrated with an example. Suppose that the underlying process we are learning generates the regular language [2] $(ab)^+$. Let the data consist of the two samples *ab* and *abab*. The sequence of merges performed by the algorithm are illustrated in Figure 2. All transitions between states without specific numbers have probability 1, and the output symbols appearing above the states have probability 1. Starting with the initial model M_0 which replicates the data, successive pairs of states are merged. In this example, the criterion for which states to merge is to choose the pair that results in a model with the smallest decrease in sample likelihood over all candidate merges. The logarithm of the likelihood (base 10) appears to the right of each model M_i. The notation $M_i : q_j; q_k$ indicates that model M_i was the result of merging states q_j and q_k (highlighted in turquoise) in model M_{i-1}. Merging two states requires removing the old states (the combined state is renumbered with the smaller of the indices of the merged pair), redirecting transitions to

[2]For an introduction to regular languages, see [Aho *et al.*, 1974]

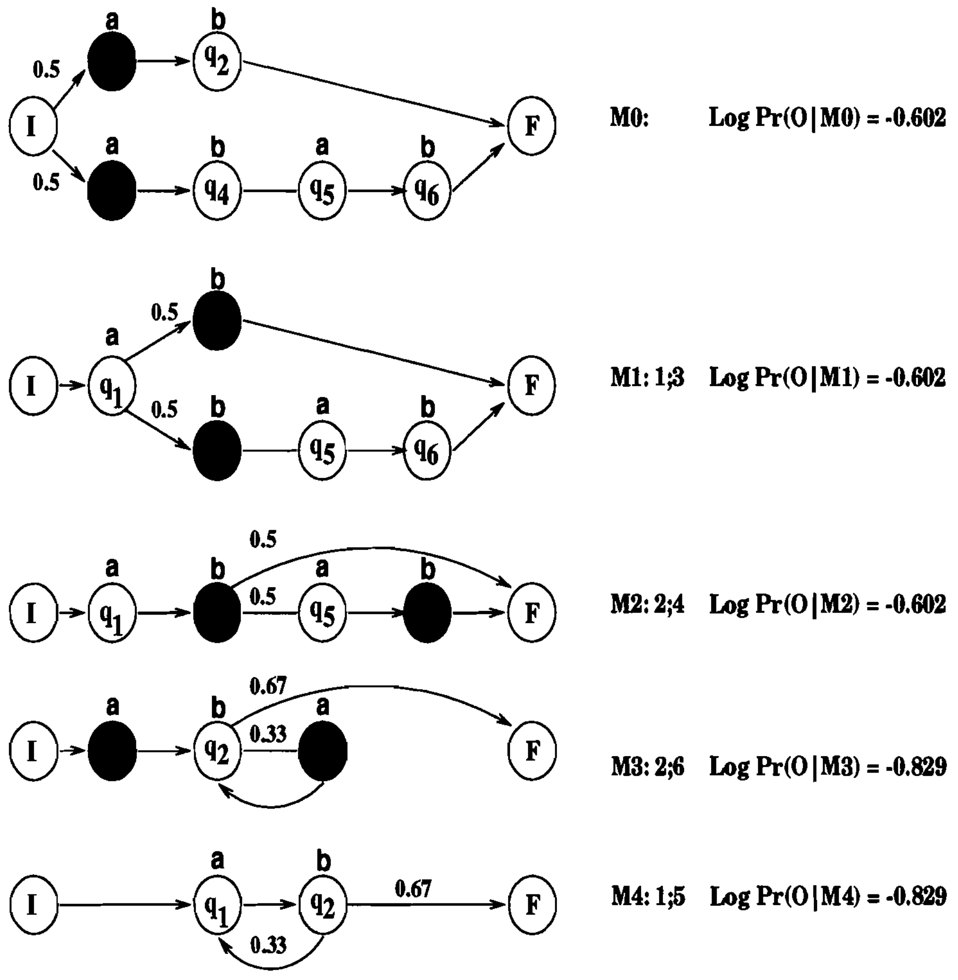

a
b
0.5
q2
I
F
M0: Log Pr(O|M0) = -0.602
a b a b
0.5
q4 q5 q6
b
0.5
a
q1
M1: 1;3 Log Pr(O|M1) = -0.602
0.5 b a b
q5 q6
0.5
a b a b
0.5
q1 q5
M2: 2;4 Log Pr(O|M2) = -0.602
0.67
a b a
0.33
q2
M3: 2;6 Log Pr(O|M3) = -0.829
a b
q1 q2
0.67
0.33
M4: 1;5 Log Pr(O|M4) = -0.829

Figure 2: An example of Bayesian Model Merging

and from the merged pair, and readjusting the transition and emission probabilities for the combined state as a weighted mixture of the individual distributions for each state in the merged pair.

In M_0, merging states q_1 and q_3 suffers no likelihood decrease, resulting in the model M_1. The second merge also does not cause a decrease. Note that the first two merges in Figure 2 could be performed in the reverse order and incur the same drop in likelihood. The model M_4 is the minimal model generating the language $(ab)^+$, but the merging could potentially continue, to yield a model generating the language $\{ab\}^+$. However, if we look at the drop of likelihood resulting from the merge, we see a drastic reduction of 3 orders of magnitude, from -0.829 to -3.465. Since we want the merging process to stop when we have achieved "desirable" generalization, we wish to disallow the merging of those final two states. One suggestion is to provide a threshold and only allow merges whose resulting drop in likelihood is below the threshold. However, determining the threshold may be difficult as it may depend heavily on the domain, the amount of available data, *etc.* The actual algorithm uses a more sophisticated technique.

The criterion for choosing which states to merge tries to sacrifice as little of the sample likelihood as possible while maintaining a bias toward smaller models capable of generalization. This tradeoff can be captured in a Bayesian formulation. From Bayes' rule, the posterior model probability $\Pr(M|O)$ of the model M given the data D is proportional to the model prior $\Pr(M)$ and the sample likelihood $\Pr(O|M)$. Smaller models will have a higher prior probability, which can outweigh the drop in sample likelihood resulting from the merging process, provided that the merging is conservative. Each step in the merging process yields a new model and the objective of the algorithm becomes finding the sequence of merges that results in the model with the maximum posterior probability.

To avoid searching the whole space, the algorithm adopts a greedy strategy of choosing, at each step, the pair of states to merge in a given model that yields a new model with the maximum posterior probability from among the set of models resulting from each potential merge. The process stops when all potential merges within a given model lead to a decrease in the posterior. A number of further optimizations are employed to ensure that each update does not require a global recomputation of the probabilities; rather, the computation is localized to those states affected by each potential merge. Details about these approximations and about the choice of model priors can be found in the technical report [Stolcke and Omohundro, 1994].

Note that the number of states n in the initial model is proportional to the number of data points in the training sample. Each step in the algorithm, therefore, has to consider $O(n^2)$ pairs of candidate merges. To deal with the complexity, the algorithm can be made on-line, where samples are incorporated a few at a time to reduce the number of states in the model and save on the computation required at each step.

Since the MM algorithm uses a best-first search for states to merge and halts as soon as there is a decrease in the posterior, it is only guaranteed to find a local maximum. One strategy to overcome this problem is to continue from that point for a fixed number of steps of *lookahead* to see whether the decrease was simply temporary. Another strategy used by the algorithm concerns the trade off between generalization and data fit. Generalization is driven by maximizing $\Pr(M)$, whereas data fit is driven by maximizing $\Pr(O|M)$. In practice, it is desirable to have a parameter which can control the balance between these two factors. To obtain control over generalization, we include a *prior weight* λ and maximize the sum $\lambda \log \Pr(M) + \log \Pr(O|M)$, which is simply the logarithmic version of Bayes' rule where the model prior is weighted by λ. For $\lambda > 1$ the algorithm will stop merging later, and earlier for $\lambda < 1$.

The performance of the algorithm is sensitive to the values for the number of *lookahead* steps, as well as the value for the λ parameter. To achieve the best results, these values must be adjusted by hand from trial and error; by experimenting with a number of settings and looking at the results of the algorithm with a particular parameter setting. More about the parameter settings appear in our discussion of the experimental results.

3 Empirical Evaluation

We tested the algorithms on two domains: regular languages, such as the example of the previous section, and the robot domain mentioned in the introduction. Both of the domains were simulated, in that we had the target model to generate data and to directly compare against the learned models. The benefit of using artificial data from a known source is that it allows us to test the algorithms over different sample sizes, and to investigate the effects of varying the parameters of the algorithms. An empirical measure of error can also be calculated since we know the target distributions.

Our original intent was to first experiment with these small domains, to get an idea of where the algorithms worked or failed, and proceed to larger problems. Unfortunately a number of issues prevented such an extended investigation and these problems are covered in depth in the following subsections.

3.1 Regular Languages

HMMs can be viewed from the formal language perspective as a stochastic generalization of nondeterministic finite automata. A model induces a probability distribution over the strings of a regular language. In fact, it can be shown that HMMs accept exactly the class of regular languages [Dean *et al.*, 1996b]. Thus, one natural application of HMMs is the induction of the automaton for a stochastic regular language. Although work exists by [Angluin, 1987, Rivest and Schapire, 1989] for inferring the automaton of a *deterministic* regular language, we might instead choose to use techniques for learning HMMs if the data is noisy.

The models learned for regular languages are often non-ergodic, *i.e.,* every state is not reachable from every other state in a finite number of steps. All the examples we considered were generated by a non-ergodic model. In these models, there is a definite notion of a final (absorbing) state from which no other states may be visited. Hence, learning such a model requires that the training data sufficiently *covers* the model, by exercising each transition and emission probability at least once. For this reason, our training data consist of several sequences, rather than one long sequence, as is sufficient for ergodic models.

One of our objectives in considering regular languages was to recreate the results of [Stolcke and Omohundro, 1994]. Their experiments showed MM to be superior to BW on two examples since they were able in each case to induce the structure of the minimal HMM generating the languages using MM. We mentioned the sensitivity of MM to the setting of the lookahead and λ parameters and our hope was to use this domain to investigate how difficult it was to determine the correct settings for these parameters in order to induce the correct model. Our investigation showed that, despite the claims of the creators, finding the right parameter setting was an art, even for such a simple domain.

The performance measure used to evaluate the algorithms was the log likelihood of a test set given each model. This quantity is proportional to the negative cross-entropy between the learned model and the target model, which reaches a minimum when the distributions are identical. Intuitively, this measure indicates how closely the learned model approximates the target distribution.

Since our experiments closely mirror those of [Stolcke and Omohundro, 1994], we do not provide specific numbers. Rather, we give a qualitative analysis of the two algorithms on three languages in particular. The general method of experimentation was to generate a random training sample of sequences, typically 10 to 20, and train the algorithms from these samples. The learned models were then tested on a sample of 20-50 sequences, and compared on the basis of whether they found the correct structure, whether they overgeneralize or overfit the data (which could be roughly estimated by inspection), and whether they performed well under the sample likelihood measure. The languages we considered include the language $(02^*0) \cup (01^*0)$ which appears in the original study, the language $(01)^+$, and the language 1^+010^+.

3.1.1 The Language $(02^*0) \cup (01^*0)$

The language $(02^*0) \cup (01^*0)$ provides an interesting test case since there is a dependency between the first and last symbol, separated by an arbitrary number of intervening symbols. This long term dependency seems to give MM an advantage, since the character of the algorithm is heavily data driven, whereas the performance of BW depends strongly on the initial parameter settings.

As was the case in the original paper, MM found the correct structure for the minimal HMM generating the language $(02^+0) \cup (01^+0)$. The models found by BW tended to overgeneralize to the language $(0 \cup 1)2^*(0 \cup 1)$, missing the dependency between first and final symbols. In terms of log likelihood, MM was found superior to BW.

The interesting aspect of this experiment involved varying the parameter settings of the MM algorithm. We found that there were large ranges of the λ parameter, in particular, which yielded the same model. The correct model was found in the range of $\lambda = 0.23$ to $\lambda = 0.03$, where values above 0.23 resulted in a model which overfit the data, and values below 0.03 resulted in a model that overgeneralized to $(0 \cup 1)2^*(0 \cup 1)$. With this example, it appeared that finding the correct setting was relatively easy. There appeared to be a nice correlation between the λ parameter and the specificity or generality of the model. However, experience with other regular languages revealed that not all languages demonstrate such a nice correlation, suggesting that there might be certain properties of this particular language that makes the parameters easier to estimate.

3.1.2 The Language $(01)^+$

The language $(01)^+$ was chosen because it is proven for BW that a uniform distribution for all parameters in a 2-state, 2-observation model is a local maximum [Dean *et al.*, 1996a]. Starting from a uniform (or nearly so) distribution causes the BW algorithm to learn a uniform model, *i.e.,* all probabilities are 0.5. Starting from almost any other distribution leads the algorithm to the optimal model which has deterministic transitions between a state generating the symbol 1 and a state generating a 0.

MM, the other hand, also proved problematic. Here the correct structure could be found, but the setting of the λ parameter was less straightforward than in the previous example. Our experiments showed that different training data required different λs in order for MM to find the correct model. It no longer was the case that broad ranges of λ would lead the algorithm to the solution, regardless of the training sets used.

3.1.3 The Language 1^+010^+

The language 1^+010^+ was selected to test whether an algorithm was likely to overgeneralize to the language $(1^+0^+)^+$. Experimental results show that BW very often did just that. MM was able to induce the correct HMM with four states plus a final state for $\lambda \geq 1$. Values for $\lambda < 1$ tended to yield models with a high number of states, indicating merging was stopped too early and the model overfit the data. Furthermore, it was often observed that, unlike the first example of a regular language, there was not a continuum of λ values, where higher values were correlated with data overfit, and lower values with overgeneralization. Rather, a lower value exhibited overfit, a higher value found the correct structure, and the intermediate value overgeneralized. One possible explanation is that because the sequence of

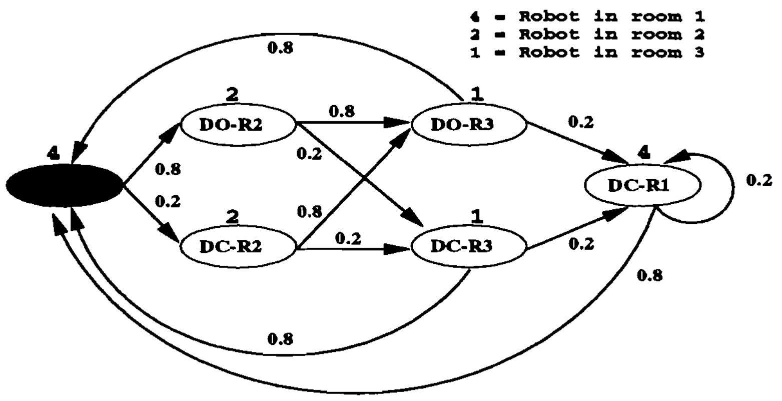


Figure 3: Actual Model for the Robot Domain

merges differs depending on the choice of λ, a particular choice of merges could prematurely collapse the structure, from which point the algorithm drastically overgeneralizes.

3.1.4 General Conclusions

The results on regular languages are viewed with mixed feelings. Experimenting with the λ parameter proved to be less straightforward than was reported in [Stolcke and Omohundro, 1994]. The authors claim that the parameters seem robust to sample size and distributions, but our results argued for the contrary. An additional claim of the authors states that the λ parameter can be estimated by starting with a high value, learning a model, and if the learned model is too specific, we could just feed the learned model back into the algorithm, using it as a starting point, and continue to merge using a lower value. Our experience shows, not only that there may not be a smooth correlation between λ values and overfit/overgeneralization, but that the precise sequence of merges changes with each λ value. Consequently, the results of such a procedure would not be the same as if the lower λ value was used from the start of the learning process.

3.2 Robot Domain

The robot domain is the same example used in the introduction. A robot can be in one of three rooms, where a door separates room 1 and room 2. The robot moves from room 1 to room 2 to room 3 and then back to room 1 in a continuous cycle. If the robot is in room 1 and the door from room 1 to room 2 is closed, the robot remains in room 1 until the door opens. The door is open 80% of the time.

The model used to generate the data has 6 actual states as shown in Figure 3, where the states are described in terms of the robot's location and the status of the door. Observations are made of the robot's location only, so the observations are a deterministic function of the state but the actual state is hidden. The output symbols are "4", "2", and "1", corresponding to whether the robot is in room 1, 2, or 3, respectively. The observation made in each state appears in red. As noted in the introduction, the dynamics can also be described using only 4 states, since the door does not affect the transition from room 2 to room 3. The 4-state model appears in Figure 4.

The robot domain was chosen as a test domain because of its structure and simplicity,

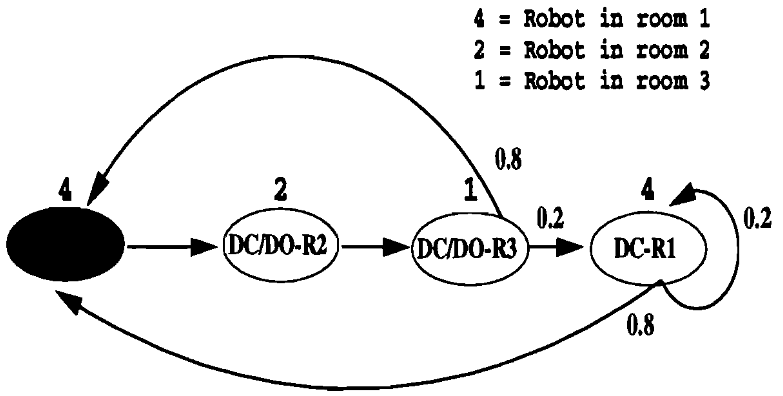


Figure 4: Consolidated Model for the Robot Domain

giving us a slightly more complex example but one that was still understandable. One nice aspect about this domain is that all of the component functions governing transitions and emissions are deterministic, except the one governing the probability of the door being open. We hoped that the determinism would make this an easy domain to learn.

The dynamics of this domain can be described by the regular language $(4214^*)^+$ which differs from the previous examples in that the generating model is ergodic. By this we mean that there is no notion of a final (absorbing) state, and a single long data sequence, that exercises the transitions and emissions often enough, is sufficient for learning the model. These two facts become problematic for the MM algorithm. The following paragraphs discuss each in turn.

Recall that the performance of MM is dominated by the need to examine $O(n^2)$ pairs of states to merge at each step, where n is the number of states in the model. Though the algorithm can be made on-line, not much is gained here since we now must use long data sequences to properly estimate the model parameters. Granted, the robot model only has a few number of states, so we could use several medium length sequences. However, for the models we are ultimately interested in, with large numbers of states, the sequences must be long enough to allow the transitions and emissions to be visited sufficiently often.

The consequence of this requirement manifests itself quite dramatically in the case of MM. The performance suffers tremendously when the number of states in a model exceeds 100. To give a rough idea of the performance lag, learning from an initial model with, say, 200 states requires approximately 2 days of computation. Our original hope was to investigate more complex domains than the simple ones of regular language induction and the robot model. However, a more complex model implies higher connectivity, and a larger state space, which means longer training sequences are necessary, rendering such an indepth investigation infeasible. Even within a single domain, it is hard to evaluate the performance of MM given that the algorithm is so sensitive to the lookahead and λ parameter settings and experimenting with the values requires an immense amount of time. Consequently, our reported results for this domain assume a single "vanilla" parameter setting of 0 lookahead steps and $\lambda = 1$. Additional tests were performed in a few interesting cases.

The second problem we encountered with using MM in the robot domain concerns the algorithm's assumption of a final state. The definition assumes the data was generated by a non-ergodic model where there is an explicit notion of a final state. It is not clear how best to apply the MM algorithm to domains without final states. If we simply ran the algorithm directly on the data sequences, the choices made by the merging process could be arbitrary for the states connected to the final state, since the algorithm could erroneously discover structure that was not inherent in the original process. Two modifications to the algorithm were attempted. Figure 5 is used to facilitate our discussion of both approaches. We refer to the shaded states in the figure as "penultimate" states.

- **Strategy 1:** We apply the algorithm directly, assuming a final state, to obtain a model M. Then we adjust the probabilities of the penultimate states in M to obtain the model M' as follows.
  - For each penultimate state i in M with a single outgoing transition to the final state (shaded blue in Figure 5), replace i by a state j, in M', having a random transition distribution over all states.
  - For each penultimate state k in M with multiple transitions to states other than the final state (shaded yellow in Figure 5), replace k by a state l, in M', where the weight associated with the transition from k to the final state is distributed equally among the other non-final state transitions from k.
  - Remove the final state from M.

  Using this method, we allow the merging process to "suggest" the states to which the penultimate states should transition. The approach, therefore, accounts for the possibility that any structure found by the algorithm is *not* arbitrary, *i.e.,* there must have been enough evidence in the data to motivate the merges involving the penultimate states. We refer to this approach as Strategy 1 in the remainder of the paper.

- **Strategy 2:** We apply the algorithm, assuming the transition distributions from the penultimate states in the initial model M_0 are *undefined.* By this we mean the penultimate states have no outgoing transitions. The merges made will rely on the output characteristics only. Any penultimate state which remains unmerged in the resulting model M will be assigned a random transition distribution, as described above. Using this method, we consider the possibility that the transition to the final state is an arbitrary one, an artifact of our desire to produce a data sequence of a particular length. The motivation behind this approach is that we want to avoid imposing a transition distribution on a state for which we have no information, rather, we wish to be cautious and leave the transition probabilities undefined. This strategy is henceforth referred to as Strategy 2.

Admittedly, both of these approaches are difficult to justify formally, yet any other approach seems equally problematic. Experiments with the robot domain show the second modification, which involves leaving the transitions undefined for penultimate states, to yield models with better performance than the first in most cases.

3.2.1 Performance Measures

As was the case in the experiments with regular languages, the target model generating the data is available. Therefore, we can again use the model to compute an empirical measure

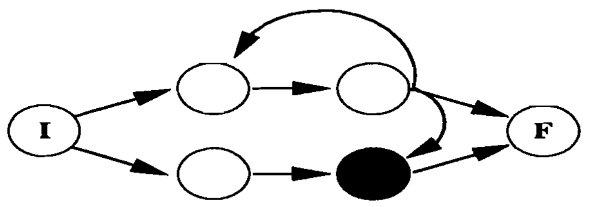


Figure 5: Dealing with the Final State in MM

of the error for the learned distributions, in terms of the target. Recall that our measure for performance in the previous experiments involved the log likelihood of test data, given the model. One important aspect to note is that the use of log likelihood only indicates how close the learned model approximates the target distribution. It does *not* indicate directly whether the learned model overfit or overgeneralized. Our previous experiments were less concerned with this fact since we knew the minimal model. In the robot domain, it is less clear whether or not we consider the 6- or 4-state model the correct model. Also, the motivation for developing a model at all for the robot domain is less one of recognition or acceptance of a language, as in the previous case, and more one of obtaining a good approximation of a model to use as a proxy for the real model, or for predicting future behavior of the underlying process. For this reason, we introduce two alternative performance measures which are symmetric, *i.e.*, they reward a learned model which nicely approximates the target and penalize for overfitting or overgeneralization.

Since an HMM induces a probability distribution over data sequences, one way of comparing HMMs is to compare their corresponding probability distributions. The most commonly used measure for this purpose is the Kullback-Leibler divergence [Kullback and Leibler, 1951]. The Kullback-Leibler divergence is related to log likelihood since the higher the likelihood, the lower the Kullback-Leibler distance between the induced distributions of the target model and the learned model. As argued previously, we instead prefer symmetric measures since we are mostly interested in how the learned model supports prediction of future observations from past observations, rather than in how well the learned model captures the hidden dynamics (except insofar as it contributes to prediction).

The following subsections describe the two performance measures we used in our experiments. The first measure captures how close two models are in terms of their induced distributions. The second measure compares models based on their predictions. Both of these measures require sampling the space of possible sequences of observations.

3.2.2 Distribution based Measure

This measure is based on the information-theoretic Kullback-Leibler divergence for a discrete domain. It is adapted to be symmetric, and is due to Juang and Rabiner [1985].

Since a detailed description is given in both [Juang and Rabiner, 1985] and [Rabiner, 1989], we only briefly review it here. In the rest of the paper this measure is denoted as *JR*.

We begin with some preliminary definitions.

Let $P = \{p_1, \dots, p_n\}$ and $Q = \{q_1, \dots, q_n\}$ be discrete distributions, *i.e.*, $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i = 1$ and $0 \leq p_i, q_i \leq 1$.

The *Kullback-Leibler divergence* [Kullback and Leibler, 1951] of Q with respect to P is

$$D(P||Q) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} \quad .$$

Since there are infinitely many sequences, applying KL directly to sequences distributions is not feasible. Hence a sampled version of it is used.

Let M_1 and M_2 be models. Let O_1, O_2 be (representative) sequences of length T, generated by M_1, M_2 respectively.

The asymmetric sampled distance is defined as:

$$D(M_1, M_2) = \frac{1}{T} \log \frac{\Pr(O_1|M_1)}{\Pr(O_1|M_2)}$$

The symmetrized version of it is:

$$D_s(M_1, M_2) = \frac{D(M_1, M_2) + D(M_2, M_1)}{2}$$

For non-ergodic models, there is a need to use multiple sequences from each model, and calculate their probabilities as a conjunction of independent sequences, which corresponds to a product of the separate probabilities.

We note that there are two drawbacks to this measure for our purposes:

- It is unbounded, since the distributions are over arbitrary sequences. Hence it is hard to conclude from a single result if two models are far apart or rather close to each other. We only know that a value of 0 indicates a perfect match between the learned and the true model distributions.
- It compares the overall distributions corresponding to the models, and does not concentrate on comparing predictions based on the models.

We have developed another information-theoretic measure that addresses these two concerns.

3.2.3 Prediction based Measure

Let $\Pr(v|O, M)$ denote the probability that the next observation is v given the sequence of observations O and the model M. As before, let M_1 and M_2 be models, and let P, Q be distributions. However, P and Q in this case are distributions of the form $\Pr(v|O, M)$ (rather than $\Pr(O|M)$).We note that there is a fixed number of observations over which $\Pr(v|O, M)$ is defined. Hence, the KL measure $D(P||Q)$ is easy to compute.

We define the *maximum Kullback-Leibler divergence* with respect to P as

$$D_{\max}(P) = \max_Q D(P||Q) \quad .$$

Due to the finite number of observations, we can prove that $D_{\max}(P)$ exists and can be computed in time quadratic in m (where m is the number of possible observations).

We define a performance measure $D(M_1, M_2)$ that is symmetric ($D(M_1, M_2) = D(M_2, M_1)$), positive, and bounded ($0 \leq D(M_1, M_2) \leq 1$). Let O_i be the set of all sequences of a fixed

Set	JR					KLP				
No.	BW-4	BW-6	BW-16	MM-1	MM-2	BW-4	BW-6	BW-16	MM-1	MM-2
1	2.10e-0	1.47e-3	9.89e-2	6.70e-1	1.26e-1	1.25e-1	1.88e-6	9.50e-5	5.00e-2	2.50e-2
2	1.74e-0	8.49e-2	1.80e-2	1.40e-0	3.60e-1	7.50e-2	1.95e-5	5.47e-5	1.76e-1	1.00e-1
3	6.99e-1	1.02e-2	6.50e-1	1.06e-0	3.54e-1	7.50e-2	9.89e-7	6.70e-4	1.00e-1	1.50e-1
4	7.78e-5	6.40e-2	1.12e-1	1.75e-2	5.24e0	7.18e-7	1.12e-6	1.27e-5	4.86e-6	3.50e-1

Table 1: Performance on robot domain.

length generated by M_i. Let O be a set of pairs drawn randomly from $O_1 \times O_2$. Let $P_i(O)$ be the distribution governing the next observation given O and M_i, *i.e.*, $P_i(O) = \{\Pr(v_1|O, M_i), \ldots, \Pr(v_m|O, M_i)\}$; then

$$D(M_1, M_2) = \frac{1}{|O|} \sum_{\langle O_1, O_2 \rangle \in O} \frac{1}{2} \left[\frac{D(P_1(O_1)||P_2(O_1))}{D_{\max}(P_1(O_1))} + \frac{D(P_2(O_2)||P_1(O_2))}{D_{\max}(P_2(O_2))} \right] .$$

We refer to this measure as *KLP*. Since it is a normalized measure, a value of 0 corresponds to an exact match of the next observation distributions, while 1 indicates that the two distributions are as far apart as possible.

3.2.4 Experimental Results

In this set of experiments, the algorithms learned from 4 data sets – each containing 10 sequences of length 50 – to yield 4 different models. For the MM algorithm, the entire data set was used in batch mode and the parameter setting was $\lambda = 1$ with 0 lookahead steps. Results are reported using both strategies for dealing with the final state. For the BW algorithm, we varied the number of initial states in the model, using 4, 6, and 16 states. The initial distributions were random and the algorithm was run until the probabilities in the matrices were not changed by more than 1.0e-5. All measures were computed on 20 pairs of test sequences of length 50. Table 1 shows the results of each algorithm on each data set, according to each performance measure.

From the table, we can see that the BW models with 6 states generally outperform all others, with the exception of the 4 state model trained on data set 4. The reason the model derived from the fourth data set performs so well under both measures is that it looks very much like the model in Figure 4. In many cases, the models learned by both algorithms overgeneralized to (roughly) the language $(4{*}21)^+$. Such a generalization is quite plausible since the fact that a single "4" is observed at the start of each sequence may be mistaken as a mere statistical anomaly.

Strategy 2 for dealing with the final state in MM appears to perform better than Strategy 1, except on the model trained on data set 4. The poor performance of MM relative to BW can be explained by looking at the number of states in each of the MM models. For each respective data set, the number of states under Strategy 1 is 9, 27, 27, and 16. For Strategy 2, the numbers are 31, 25, 6, 57 respectively. Looking at the resulting models shows that the merging process was stopped too early, leaving long "strings" of the original state sequences unmerged with any other state. This indicates that the models overfit the data and were subsequently penalized under the performance measures.

To test how the λ parameter affected generalization in this domain, we trained a model using MM with Strategy 2 and $\lambda = 2$ on training set 4 and the resulting model appears

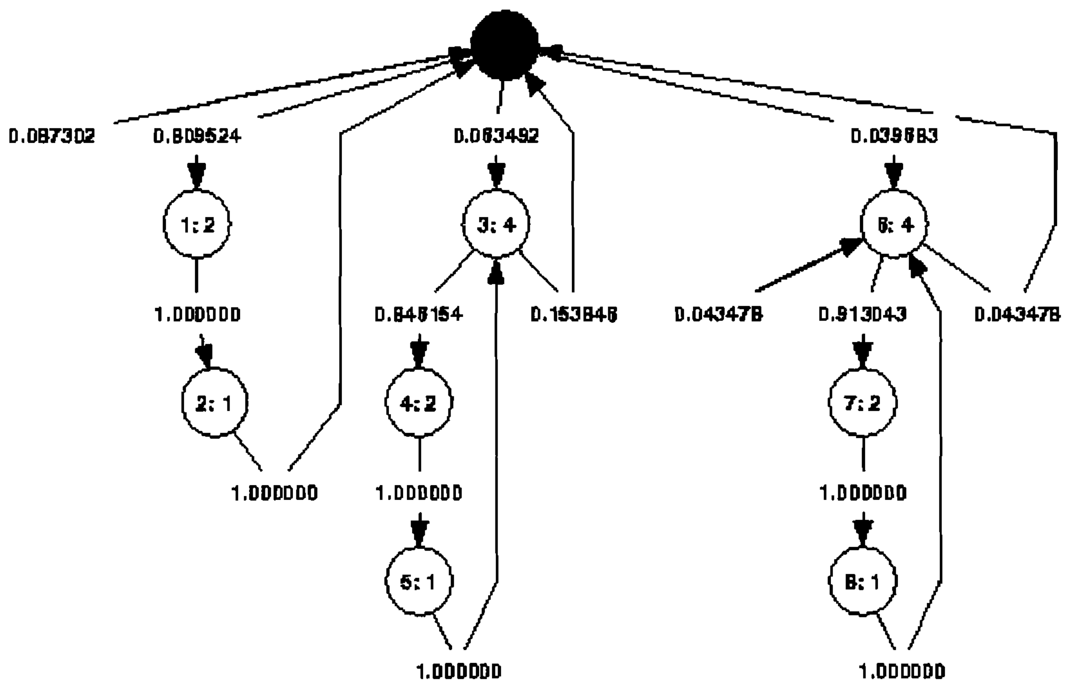


Figure 6: MM-2 Trained on Data Set 4 ($\lambda = 2$). The shaded circle indicates the start state.

in Figure 6. The performance was 1.75e-0 for the JR measure and 1.25e-1 for the KLP. Note that the self-transition on state 0 indicates that the model also overgeneralizes, since it can generate the observation sequence $4, 4, 2, 1, 4, \ldots$ which was not possible in the original model. In fact, examining the results of the tests show that the model was often penalized for generating such a sequence.

The notable exception for MM is the model learned on data set 3 with Strategy 2, which appears to do better than both the 4-state and 16-state BW counterparts under the JR measure. The model, shown in Figure 7, only has six states, which contributes to its success at generalization. However, note that this model will also be penalized for overgeneralizing by generating the sequence $4, 4, 2, 1, 4, \ldots$.

One final note concerns an earlier observation about why we chose to use symmetric measures. If we only consider the performance of MM relative to BW in terms of sample log likelihood, we get a different picture than the one evident in Table 1. We tried an experiment where the 16 state BW models were compared against the Strategy 2 MM models using log likelihood on three test sets of 20 sequences of length 50. The results appear in Table 2. The stars indicate that the learned models were too specific (*i.e.*, they overfit the training samples) and failed to parse some of the samples. For instance, no path generating the sequence 4,2,1,4,4,4,4,4,2,1 existed in the model learned for data set 1 using MM. However, for the models trained on data set 2 (appearing in row 2), we see that in terms of log likelihood, MM appears to outperform BW. This means that the distribution over observation sequences induced by the MM model was closer to the target than the BW model. The difference between the symmetric and asymmetric results can be attributed to the fact that invariably

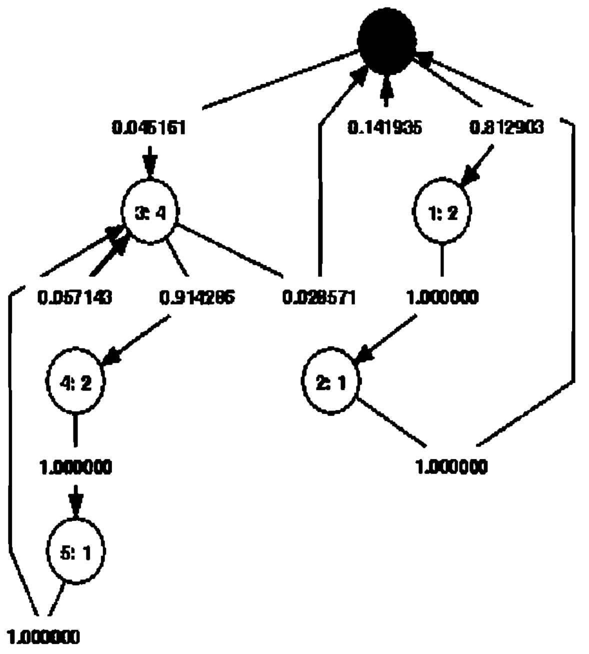


Figure 7: MM-2 Trained on Data Set 3 ($\lambda = 1$). The shaded circle indicates the start state.

Training	Test Set 1		Test Set 2		Test Set 3	
Set	BW-16	MM-2	BW-16	MM-2	BW-16	MM-2
1	-285.944	*	-187.362	*	-226.793	*
2	-257.837	-97.382	-89.554	-81.637	-169.062	-83.817
3	-770.509	-94.560	-643.741	-78.186	*	-80.225
4	-239.028	*	-167.896	*	-171.077	*

Table 2: Sample Log Likelihood on 3 Test Samples

the MM models overgeneralized since it allows the sequence $4, 4, 2, 1, 4, \ldots$.

4 General Conclusions

The general conclusions regarding the effectiveness of MM are disappointing. The possibility that an algorithm could correctly learn *both* the model topology and parameters is highly attractive. Our experience with MM is disheartening, since the algorithm failed to meet our expectations. Though the MM algorithm seems elegant and intuitive, it is much too time consuming to be practical when applied to any interesting domains. The results from the experiments with regular languages show that MM can easily be applied to domains where the sequences are short and well structured, as in words or regular expressions, but even this comes at the price of having to estimate the algorithm's parameters. Our experience shows that finding the right values for the algorithm is more difficult than the reports of [Stolcke and Omohundro, 1994].

Another major problem is in trying to apply MM to domains where the underlying process is ergodic. There appears to be no natural extension of the algorithm to deal with domains where there is no distinct notion of a final or absorbing state. The two approaches we attempted resulted in models which approximate the target distributions well but suffered from overfit and overgeneralization. It is not clear what other alternatives we might consider.

We offer one key insight into why MM performs so poorly in terms of the models it induces. The algorithm begins with a model where the number of states is exactly proportional to the number of data points. At each state, we are attempting to learn a large number of parameters based on a *single* observation. In fact, this is precisely the case following the first several merges, since we do not have enough information at each state to reliably estimate what the parameters should be. The problem is compounded by the fact that merges made cannot be undone, so the algorithm commits to a sequence of merges that could turn out to be detrimental to the quality of the final resulting model. One strategy to overcome this problem might be to explore a sequence of k merges in the initial model instead of a single merge, gradually reducing the horizon depth as the total number of merges increases to the point where we can put more trust in the algorithm's ability to estimate the model parameters. Note that this is not the same as the lookahead procedure discussed earlier, since that mechanism is only invoked after an initial drop in the posterior probability.

In this paper we attempted to apply HMMs to the task of learning model of dynamical systems. Specifically, we investigated the performance of two algorithms for learning HMMs – Baum-Welch and Bayesian Model Merging – on several examples of simulated domains. We found that Model Merging suffers a number of drawbacks, despite prior claims of its effectiveness. The algorithm is highly sensitive to a number of parameters that are hard to estimate. Although the algorithm is designed to learn models from the class of stochastic regular languages, we also found that it is not practical for modelling arbitrarily long data sequences or large state spaces. The performance results suggest that Model Merging yields models that provide good approximations to the underlying model of the system in terms of recognizing sequences of observations from the target model with high probability. However, the learned models frequently fail by overgeneralizing, a harmful characteristic for applications where the interest is in prediction.

A further goal of this work was to provide additional evaluation of the two algorithms, since there is little literature about Bayesian Model Merging. The work that does exist is that of the algorithm's creators so this investigation was meant to be a less biased look into the applicability of the algorithm. Our hope is that the reader can learn from our experience when trying to learn models for dynamical systems.

References

[Abe and Warmuth, 1992] Abe, N. and Warmuth, M. 1992. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning* 9:205–260.

[Aho *et al.*, 1974] Aho, Alfred V.; Hopcroft, John E.; and Ullman, Jeffrey D. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts.

[Angluin, 1987] Angluin, Dana 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75:87–106.

[Baum *et al.*, 1970] Baum, L. C.; Petrie, T.; Soules, G; and Weiss, N. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions in Markov chains. *Annals of Mathematical Statistics* 41(1):164–171.

[Dean *et al.*, 1996a] Dean, Thomas; Kaelbling, Leslie Pack; Kim, Kee-Eung; Leach, Sonia; and Shatkay, Hagit 1996a. Exploiting structure in learning to predict the outputs of dynamical systems. Technical Report CS-96-**, Brown University Department of Computer Science.

[Dean *et al.*, 1996b] Dean, Thomas; Leach, Sonia; and Shatkay, Hagit 1996b. Graphical models for learning dynamical systems. Technical Report CS-96-**, Brown University Department of Computer Science.

[Dempster *et al.*, 1977] Dempster, A.; Laird, N.; and Rubin, D. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* 39 (Series B):1–38.

[Juang and Rabiner, 1985] Juang, B. H. and Rabiner, L. R. 1985. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal* 64(2):391–408.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. 1951. On information and sufficiency. *Annals of Mathematical Statistics* 22(1):79–86.

[Rabiner, 1989] Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

[Rivest and Schapire, 1989] Rivest, Ronald L. and Schapire, Robert E. 1989. Inference of finite automata using homing sequences. In *Proceedings of the Twenty First Annual ACM Symposium on Theoretical Computing.* 411–420.

[Stolcke and Omohundro, 1993] Stolcke, Andreas and Omohundro, Stephen 1993. Hidden Markov model induction by Bayesian model merging. In Hanson, S. J.; Cowan, J. D.; and Giles, C. L., editors 1993, *Advances in Neural Information Processing 5.* Morgan Kaufmann, San Francisco, California.

[Stolcke and Omohundro, 1994] Stolcke, Andreas and Omohundro, Stephen 1994. Best-first model merging for hidden Markov model induction. Technical Report TR-94-003, International Computer Science Institute, Berkeley, California.